Complete all problems.
You are not permitted to look at solutions of previous year assignments. You can work together in groups, but all solutions must be written up individually. If you get information from sources other than the course notes and slides, please cite the information, even if from Wikipedia or a textbook.
Please provide a simple illustration of your solution, more than just an answer.

**Problem 1: Binary versus q-ary Codes (10pt)**

In class we showed a lower bound on the number of code bits required to fix $s$ bit-errors for a binary linear code with $k$ message bits. In particular, by arguing that every codeword requires a ball of radius $(d-1)/2$ around it we derived the formula

$$n \geq k + \log \left( 1 + \sum_{i=1}^{s} \binom{n}{i} \right).$$

1. For any integer $q \geq 2$, give an expression for the number of words in $[q]^n$ that are within Hamming distance $s$ of some fixed word.

$$1 + \sum_{i=0}^{s} \binom{n}{i}(q-1)^i$$

1. Use this to give an upper bound for the rate of $q$-ary codes.

$$n \geq k + \log_q \left( 1 + \sum_{i=0}^{s} \binom{n}{i}(q-1)^i \right)$$

**Problem 2: A Non-Algorithmic Construction of Good Codes (10pt)**

In lecture, we argued that for any point $x$ on the $n$-dimensional hypercube, the number of hypercube points within distance $D$ of $x$ is $\text{Vol}_n(D) := \sum_{i=0}^{D} \binom{n}{i}$. One can show that for $D \leq n/2$,

$$\log_2 \text{Vol}_n(D) \approx n H(D/n),$$

where $H(p) := p \log_2(1/p) + (1-p) \log_2(1/(1-p))$ is the binary entropy function you’ve seen earlier in the course.

1. In other words, the Hamming lower bound says that the rate $k/n$ of any binary code can be at most $\approx 1 - H((d-1)/2)/n$. (Nothing to do here.)

2. Consider the following greedy algorithm to construct a distance-$d$ code. Start with $C = \emptyset$. Pick any vector from $\{0, 1\}^n$ and add it to $C$. Now, as long as there exists some
vector that is at distance at least \( d \) from every codeword in \( C \), pick one of these and add it to \( C \). Show that the code \( C \) you construct has rate
\[
\frac{k}{n} \geq 1 - \frac{\log_2 \text{Vol}_n(d-1)}{n} \approx 1 - H\left(\frac{d-1}{n}\right).
\]

Hence the lower and upper bounds differ in one having \( H\left(\left\lfloor \frac{d-1}{2n} \right\rfloor \right) \) and the other \( H\left(\frac{d-1}{n}\right) \).

Closing this gap is a big part of coding-theory research.

After the greedy procedure terminates, we claim that the set of balls of radius \( d - 1 \) around all the elements in \( C \), covers the entire space. If there did exist a point \( p \) that is not covered, then \( p \) is distance \( d \) from all elements in \( C \), thus contradicting the fact that the algorithm terminated.

Therefore, we have
\[
2^n \leq 2^k \text{Vol}_n(d-1).
\]

Taking the log of both sides,
\[
n \leq k + \log \text{Vol}_n(d-1)
\]
rearranging, we arrive at
\[
\frac{k}{n} \geq 1 - \frac{\log \text{Vol}_n(d-1)}{n},
\]
the desired result.

**Problem 3: Why are Reed-Solomon Codes Optimal?** (10pt)

Another lower bound on the rate of codes is the following: for any \((n, k, d)\_q\) code,
\[
d \leq n - k + 1 \tag{1}
\]
or equivalently, the rate of the code is at most \( 1 - \frac{d-1}{n} \). Note that Reed-Solomon codes satisfy this bound exactly, and hence are optimal (albeit only when \( q \) is large).

Prove this lower bound in (1). (Hint: what happens if you delete the first \( d - 1 \) symbols of each codeword?)

If you delete those, no two codewords collide, since their distance is at least \( d \). So we have at most \( q^{n-(d-1)} \) codewords. And we must have \( q^k \) distinct codewords. This proves the bound.

**Problem 4: Ever Wonder about ISBN?** (15pt)

The ISBN is a 10-digit codeword such as 0-471-06259-6. The first digit indicates the language (0 for English), the next group specifies the publisher (471 for Wiley); the next group forms the book number assigned by the publisher. The final “digit” is chosen to make the entire number \( x_1 \cdots x_{10} \) satisfy the single check equation: \( \sum_{i=1}^{10} (ix_i) = 0 \) (mod 11).

Note that the first 9 digits lie between 0 and 9, whereas the last “digit” can take any value between 0 and 10, where the value 10 is represented by the letter \( X \).
A. Give the parameters of this code in the \((n,k,d)\_q\) notation. (I.e., what are \(n,k,d\) and \(q\)?) \((10,9,2)_{11}\)

B. Calculate the check digit for the message 0-13-201516.

The check bit is given by the equation \(2(1) + 3(3) + 4(2) + 6(1) + 7(5) + 8(1) + 9(6) + 10(x) = 0 \pmod{11}\). Solving this gives \(x = 1\).

C. It is easy to see that the ISBN code can detect any single digit error. Show that the code can detect the transposition of any two digits (not necessarily consecutive).

Let the original codeword be \(u_1 \cdots u_{10}\), and the transposed numerals be at positions \(j\) and \(k\) respectively, \(j \neq k\). That is, \(u'_i = u_i\) for all \(i \neq j,k\), \(u'_j = u_k\), and \(u'_k = u_j\). By definition we have, \(\sum_i i u_i = 0 \pmod{11}\). Assume that the transposition is not detected. Then, \(\sum_i i u'_i = 0 \pmod{11}\). Thus, \(\sum_i i u_i = \sum_i i u'_i \pmod{11}\), or, \(ju_j + ku_k = ju_k + ku_j \pmod{11}\). This implies, \((j-k)u_j = (j-k)u_k \pmod{11}\). But, \(j-k \neq 0 \pmod{11}\). Thus, we must have \(u_j = u_k \pmod{11}\), or, \(u_j = u_k\) because both the numbers are less than 11. Thus the two numbers are the same and no transposition took place!

D. The sixth digit in the code 0-621-5?157-2 was smudged. Find the missing digit.

The missing bit is given by the equation \(2(6) + 3(2) + 4(1) + 5(5) + 6(x) + 7(1) + 8(5) + 9(7) + 10(2) = 0 \pmod{11}\), or, \(6x = 10 \pmod{11}\). We get \(x = 9\).

Problem 5: Reed-Solomon Codes (10pt)

Suppose we have an inexpensive (and fast!) PCI board that implements an \(RS(255,223)\) Reed-Solomon encoder and decoder in hardware. (Assume you are working in the field \(\mathbb{F}_{256} = \mathbb{F}_{2^8}\), and hence each symbol is one byte long.) The board encodes messages with \(k = 223\) bytes, decodes received-words of \(n = 255\) bytes, and corrects up to 16 byte errors in each block. You would like to use Reed-Solomon codes to protect your data against errors as it is transmitted over a wireless communication channel.

Unfortunately, your radio experiments show that, at your transmission rate, bursts of errors tend to be longer than 16 bytes. Using the \(RS(255,223)\)-encoder/decoder as a building block, design a system that can correct up to 64 bytes of consecutive errors in a 1020-byte transmitted message, assuming that there are no other errors in the message. You must preserve the rate of the channel.

We divide the message into sequences of length 16 bits each. Let these sequences be \(S_1 S_2 S_3 \cdots\). Then, we encode \(S_1 S_3 S_9 \cdots\) using the \(RS(255,223)\) encoder. Likewise we encode \(S_2 S_6 S_{10} \cdots\), \(S_3 S_7 S_{11} \cdots\), and \(S_4 S_8 S_{12} \cdots\) using the \(RS(255,223)\) encoder. Clearly the rate is preserved. Furthermore, note that, any consecutive bit errors containing up to 64 bits are divide over the four sequences, such that each sequence contains at most 16 of them consecutively. Thus this code can correct up to 64 consecutive bit errors. This is known as interleaving, and is
commonly used, e.g., in CDs and DVDs, where scratches and other such errors can indeed be modeled as bursty.

Problem 6: Code Concatenation (10pt)

Suppose we concatenate a \((N, K, D)_{q^k}\) code with a \((n, k, d)_{q}\) code. Show that the resulting code is a code with parameters \((Nn, Kk, Dd)_{q}\).

We concatenate the codes by first encoding a message of size \(K\) in the \((N, K, D)_{q^k}\) code. Each symbol in the large alphabet is \(k\) symbols in the small alphabet, so the message is \(Kk\) symbols in the small alphabet. Then the codeword is size \(N\) in the large alphabet. We encode each of the \(N\) symbols using the \((n, k, d)_{q}\) code, which turn into \(n\) symbols each. Therefore, the final codeword is \(Nn\) symbols long.

Given two codewords in the concatenated code, their corresponding \((N, K, D)_{q^k}\) codewords must differ by at least \(D\) symbols in the large alphabet. In each of these places, there must be at least \(d\) symbols that differ in the small alphabet, since they were encoded using \((n, k, d)_{q}\). So in total, the codewords must differ by \(Dd\) symbols.

Problem 7: A Variant on LDPC codes (15pt)

Consider the following variant on LDPC codes. Like LDPC codes the code is given by a bipartite graph. But now, for each node on the right, the bits at its neighbors must form a proper Hamming code. To be concrete, each vertex on the right has degree 15, and the bits on its neighbors must form a \((15, 11, 3)\) Hamming code. Assume each vertex on the left has degree \(d = 3\), so the number of nodes on the right is \(n/5\).

1. What is the rate of this code (i.e. \(k/n\))? Each node on the right contributes 4 equations—a \((15, 11, 3)\) Hamming code has a \(4 \times 15\) parity check matrix. The total number of constraints are therefore \(4n/5\). There are \(n\) variables so after removing \(4n/5\) degrees of freedom we are left with \(n/5\) degrees of freedom (a message word will be \(n/5\) bits long). The rate is therefore \(1/5\).

2. Assuming the bipartite graph has expansion \((\alpha, \beta)\) with \(\beta = d/2 = 1.5\) prove that the code has distance at least \(\alpha n\). This is a similar argument to the one given in class for LDPC codes, but for LDPC codes we required that \(\beta > d/2\).

Consider \(l \leq \alpha n\) errors on the left. We will prove that this will be recognized as an error and hence the distance must be greater than \(\alpha n\). The nodes on the left will have at least \(3/2l\) neighbors on the right. There will be a total of \(3l\) edges coming into the right with errors on them. Therefore the average number of errors on a node on the right with a neighboring error
is at most 2. Since the average is at most 2, then at least one of the nodes on the right has no more than 2 errors (1 or 2). The hamming code will recognize this as an error.